JsonApp

Vaším úkolem je vytvořit jednoduchou knihovnu, která bude sloužit jako jednoduchý (ale obecný) konvertor JSON na objekty a zpět. Kromě samotné knihovny je dále potřeba vytvořit demonstrační příklad, který danou knihovnu bude používat k evidenci dat v souborech formátu JSON.

Výsledkem semestrální práce jsou dva propojené projekty – projekt JSON knihovny (DLL knihovna) a demonstrační aplikace (spustitelný konzolový program).

JsonLibrary

JsonLibrary představuje knihovnu, která nabízí objektové mapování jednotlivých JSON elementů.

- Knihovna musí umět pracovat se základními JSON typy: null, number, string, bool, array, object.
- U řetězců musí být schopna pracovat s escapováním pomocí zpětného lomítka minimálně pro zpětné lomítko a uvozovky. Nemusí být schopna pracovat s Unicode escape sekvencemi.
- U čísel není nutné řešit čísla zapsaná s exponentem (123e10, 1e+1, 1e-0, 0E1)
- Knihovna nemusí řešit ignorování nadbytečných bílých znaků („formátovaný json“, „pretty-printing“, znaky doplněné pouze za účelem zlepšení čitelnosti – bez strojového zpracování), pokud její vlastní serializace produkuje JSON bez těchto bílých znaků.
  - Bez bílých znaků:
    `{"alpha":{"beta":100,"gamma":200},"delta":[0,1,2,3,4,null]}`
  - S bílými znaky (jedna z možností):
    `{ "alpha" : { "beta":100, "gamma":200}, "delta": [ 0 , 1 , 2,3, 4 , null ]}`
- Knihovna musí umět převést existující řetězec na objekty (deserializace) a objekty na JSON řetězec (serializace).
- Deserializace musí umět načíst libovolný JSON (s výše uvedenými omezeními), nikoliv jen JSON použitý v demonstrační aplikaci!

- **V předepsaném API chybí několik důležitých metod, aby bylo možné výsledný JSON strom objektů přečíst včetně všech detailů – chybějící metody doplňte.**

Předepsané API (soubor api.h)

```
#include <string>

// - šablona s parametrem datového typu uložených hodnot
// - není povoleno užití STL kontejnerů ani jiných knihoven pro ukládání dat
// - realizace musí využívat dynamicky alokované pole, spojový seznam nebo jinou vhodnou Vámi
implementovanou ADS
template<typename T>
class DynamicArray {
public:
	DynamicArray();
	~DynamicArray();

	// - přidá element na konec pole
	void append(const T& element);

	// - výjimky při neplatném nebo nekorektním indexu
	const T& getElementAt(int index) const;

	// - vrací velikost (počet prvků) v poli
	int getSize() const;
};

////////////////////////////////////////////////////////////////////////////////

// - definuje pár klíč (řetězec) a hodnota (JSON hodnota) pro reprezentaci hodnot JSON objektu
class KeyValuePair {
public:
	KeyValuePair(std::string key, Value* value);

	// - vrátí klíč
	std::string getKey() const;
	// - vrátí hodnotu
	Value* getValue() const;
};

////////////////////////////////////////////////////////////////////////////////

// JSON hodnota - reprezentuje abstraktního předka pro základní datové typy v JSON (string, number,
object, array, bool, null)
class Value
{
public:
	// serializuje hodnotu do podoby JSON reprezentace
	virtual std::string serialize() const = 0;
};

////////////////////////////////////////////////////////////////////////////////

// - reprezentuje hodnotu typu JSON null
class NullValue :
	public Value
{
public:
};

////////////////////////////////////////////////////////////////////////////////

// - reprezentuje hodnotu typu JSON bool
class BoolValue :
	public Value
{
public:
	BoolValue(bool value);
```

```
	// - vrací bool hodnotu
	bool get() const;
};

//////////////////////////////////////////////////////////////////////////

// - reprezentuje hodnotu typu JSON číslo
class NumberValue :
	public Value
{
public:
	NumberValue(double value);

	// - vrací číselnou hodnotu
	double get() const;
};

//////////////////////////////////////////////////////////////////////////

// - reprezentuje hodnotu typu JSON řetězec (string)
class StringValue :
	public Value
{
public:
	StringValue(std::string value);

	// - vrací řetězcovou hodnotu
	std::string get() const;
};

//////////////////////////////////////////////////////////////////////////

// - reprezentuje hodnotu typu JSON pole
class ArrayValue :
	public Value
{
public:
	ArrayValue();
	~ArrayValue();

	// - přidá element na konec pole
	void append(Value* element);

private:
	// - atribut DynamicArray<Value*> pro uchování jednotlivých elementů v poli
};

//////////////////////////////////////////////////////////////////////////

// - reprezentuje hodnotu typu JSON objekt
class ObjectValue :
	public Value
{
public:

	ObjectValue();
	~ObjectValue();

	// - přidá klíč-element do objektu
	void append(const KeyValuePair& pair);

private:
	// - atribut DynamicArray<KeyValuePair> pro uchování jednotlivých hodnot a klíčů v objektu
```

```
};

//////////////////////////////////////////////////////////////////////////////

// - třída pro práci s JSON
class JSON
{
public:
	// - provede deserializaci řetězce na vytvořené objekty
	// - přečtu znak a rozhodnu se (uvedený postup předpokládá čtení bez přeskakování
	//   bílých znaků)
	// -- '{' - začínám číst objekt
	// -------- čtu znaky, pak musí být dvojtečka, potom volám rekurzivně deserialize();
	            následuje čárka nebo '}', podle situace se čtení opakuje
	// -- '[' - začínám číst pole
	// -------- volám rekurzivně deserialize(); následuje čárka nebo ']', podle situace se čtení
	            opakuje
	// -- '"' - začínám číst řetězec - pozor na escapované uvozovky
	// -- [-0123456789] - začínám číst číslo - načtu všechny číslice (pozor na možnou desetinnou
	            tečku)
	// -- 'n' - 'null' - načtu zbylé znaky a kontroluji, že je to opravdu ‚null‘ a ne něco jiného
	// -- 't' - 'true' - dtto
	// -- 'f' - 'false' - dtto
	// -- cokoliv jiného - vyvolávám výjimku
	// - není přípustné vracet nullptr
	// - deserializace musí být rozumně implementována - není přípustné zde napsat jednu extrémně
	     dlouhou metodu
	static Value* deserialize(const std::string& string);

	// - provede serializaci do JSON řetězce
	static std::string serialize(const Value* value);
};

//////////////////////////////////////////////////////////////////////////////
```

Ukázka použití (soubor example.cpp):

```
void main() {
	ObjectValue* ov = new ObjectValue{};
	ov->append(KeyValuePair{ "bool", new BoolValue{true} });
	ov->append(KeyValuePair{ "null", new NullValue{} });
	ov->append(KeyValuePair{ "string", new StringValue{"hello world \\ \" experiment"} });
	ov->append(KeyValuePair{ "number", new NumberValue{3.141592} });

	ArrayValue* av = new ArrayValue{};
	av->append(new NumberValue{ 0 });
	av->append(new NumberValue{ 1 });
	av->append(new StringValue{ "two" });
	av->append(new NumberValue{ 3 });
	ObjectValue* four = new ObjectValue{};
	four->append(KeyValuePair{ "four", new NumberValue{4} });
	av->append(four);
	av->append(new NumberValue{ 5 });

	ov->append(KeyValuePair{ "array", av });

	cout << ov->serialize();
}
```

Demonstrační aplikace

Demonstrační aplikace bude využívat vytvořenou JSON knihovnu a bude demonstrovat její funkcionality.

- Je nutné využít vytvořenou JSON knihovnu.
- Téma demonstrační aplikace je libovolné, ale očekává se aplikace s konkrétní funkcionalitou – není přípustné dělat obecné načítání a editaci JSON souborů.
- Načítané a ukládané struktury musí využít minimálně 1 JSON pole a 1 JSON objekt.
- Aplikace musí nabídnout základní CRUD operace.
- Aplikace musí být schopna načítat a ukládat data do souborů.

- **Odevzdaná semestrální práce musí obsahovat připravené soubory s ukázkovými daty pro jednodušší otestování funkčnosti aplikace.**

Příklady demonstračních aplikací:

- **Evidence studentů**
  - **Formát JSON souboru: [{ student01 }, { student02 } …]**
  - **Evidované hodnoty u studenta: jméno, příjmení, id, pohlaví, datum narození, pole předmětů**
- **Knihovní systém**
  - **Formát JSON souboru { "uzivatele": [seznam-uživatelů], "knihy": [seznam-knih], "vypujcky": [seznam-výpujček] }**
  - **Evidované hodnoty u uživatele: id, jméno**
  - **Evidované hodnoty u knihy: id, autor, název, rok vydání, počet dostupných kopií**
  - **Evidované hodnoty u výpůjček: id uživatele, id knihy, index kopie, datum vypůjčení**

Pokud se rozhodnete využít výše uvedené příklady, pak je doplňte o další evidované hodnoty. Není přípustné je použít přesně v uvedeném stavu.

Technické požadavky na realizaci semestrální práce

- **Připravené API je nutné zcela dodržet**. Návrh a implementace chybějících částí (zahrnující nové metody v existujících třídách, nové atributy, nové třídy, …) je zcela na Vás. Detaily vizte přiložené soubory.

- Veškerá dynamická paměť musí být **korektně dealokována**. Není přípustné nechat paměť alokovanou déle, než je to nutné. Není přípustné nechat nějakou paměť alokovanou „po skončení programu“, veškerá dyn. alokovaná paměť musí být uvolněna nejpozději v destruktorech volaných implicitně na konci main().

- Vytvoření DLL knihovny je možné shlédnout na:
  - https://github.com/MicrosoftDocs/cpp-docs/blob/master/docs/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp.md
  - https://docs.microsoft.com/cs-cz/cpp/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp?view=vs-2019

- DLL poznámky:
  - Projekt JsonLibrary musí být nastaven, že jeho výstupem je DLL knihovna.
    - C++ - preprocesor – definice: WIN32 / WIN64, _WINDLL
  - Projekt DemonstračníAplikace musí mít nastaveno:
    - C++ - preprocesor – definice: WIN32 / WIN64
    - C++ - includes – relativní cesta k H souborům knihovny (/I)
    - Linker – knihovna – název knihovny JsonLibrary.lib
    - Linker – cesty ke knihovnám – relativní cesta k JsonLibrary.lib (/LIBPATH)
    - Pokud nebudou v nastavení projektu použity relativní cesty – nepůjde projekt na jiném pc zkompilovat a semestrální práce nebude akceptována.
  - Příklad konfigurace solution a projektů je přiložen k zadání semestrální práce.

- **Není dovoleno** využívat **globální a statické proměnné**.

- **Program vypracovaný pro platformu Windows musí obsahovat projektové soubory pro Visual Studio 2017 nebo novější, pomocí kterých lze program přeložit.**

  Pro unixové platformy musí být přiložen makefile. Zdrojové kódy, které není možné přeložit (chybějící projektové/zdrojové soubory, syntaktické chyby v kódu), nebudou akceptovány.

  Před odevzdáním musí být projekt uklizen od veškerých binárních souborů (spustitelných, tmp, cache, objektových – pomocí Build – Clean solution | make clean), dále neodevzdávejte složky: „debug“, „release“, „output“, „x86“, „x64“, „.vs“. Vyčištěné zdrojové kódy včetně souborů projektu odevzdávejte ve formátu ZIP.

- **Vyžaduje se platný kód dle standardu INCITS/ISO/IEC 14882:2017 („C++ 17“). Nejsou dovolena žádná proprietární rozšíření jednotlivých kompilátorů. Není dovoleno použít neschválené externí knihovny.**

Nejzazší termín odevzdání semestrální práce je 31. 12. 2020 23:59.

Semestrální práce musí být vypracována samostatně, není přípustná žádná shoda s jinou prací. Student musí být schopen vytvořenou práci okomentovat a vysvětlit při ústní obhajobě.